USAISEC

US Army Information Systems Engineering Command
Fort Huachuca, AZ 85613-5300

(2)

U.S. ARMY INSTITUTE FOR RESEARCH
IN MANAGEMENT INFORMATION,
COMMUNICATIONS, AND COMPUTER SCIENCES

Software Development Information Supported by Typical CASE Tools

March 1991

ASQB-GI-91-014

AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0800

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188
Exp. Date: Jun 30, 1986

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS NONE
2a SECURITY CLASSIFICATION AUTHORITY N/A	3 DISTRIBUTION / AVAILABILITY OF REPORT N/A
2b DECLASSIFICATION / DOWNGRADING SCHEDULE N/A	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) ASQB-GI-91-014	5 MONITORING ORGANIZATION REPORT NUMBER(S) N/A

6a NAME OF PERFORMING ORGANIZATION Purdue University / SERC	6b OFFICE SYMBOL (if applicable)	7a NAME OF MONITORING ORGANIZATION N/A
6c ADDRESS (City, State, and ZIP Code) Department of Computer Science West LaFayette, Indiana 47907		7b ADDRESS (City, State, and Zip Code) N/A
8a NAME OF FUNDING/SPONSORING ORGANIZATION AIRMICS	8b OFFICE SYMBOL (if applicable) ASQB - GI	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c ADDRESS (City, State, and ZIP Code)
115 O'Keefe Bldg.,
Georgia Institute of Technology
Atlanta, GA 30332-0800

10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO	PROJECT NO	TASK NO.	WORK UNIT ACCESSION NO
62783A	DY10	02-04-02	

11 TITLE (Include Security Classification)
Software Development Information Supported by Typical CASE Tools (UNCLASSIFIED)

12 PERSONAL AUTHOR(S)
Dunsmore, Buster; Varnau, Steve

13a TYPE OF REPORT final report	13b TIME COVERED FROM ______ TO ______	14 DATE OF REPORT (Year, Month, Day) 1991, March, 12	15 PAGE COUNT 27

16 SUPPLEMENTARY NOTATION

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
CASE Tools; Distributed Computing Design System; DCDS; Teamwork; Excelerator; EPOS; DesignAid; SA Tools; Software engineering environment Systems; SEES; Ada Programming Support Environment; APSE

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The work described in this report was accomplished as part of the Distributed Computing Design System (DCDS) evaluation project, AIRMICS Report ASQB-GI-91-009, "Evaluation of DCDS for Meeting the Data Collection Requirements for Software Specification, Development, and Support". The DCDS evaluation is outlined in the succeeding paragraph. This report augments the DCDS report by identifying the data collection requirements for a fully flexible CASE environment.

The DCDS evaluation technical report consists of five separate but related reports which evaluate the Distributed Computing Design System (DCDS). DCDS was developed by TRW as a software development environment for real-time, distributed systems. The principal investigator evaluated DCDS in terms of: a) its data collection requirements, b) its software development information completeness, c) its usability, d) how it compares to five commercially available CASE tools, and e) its suitability as an Ada Programming Support Environment (APSE).

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS		21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Howard C "Butch" Higley	22b TELEPHONE (Include Area Code) (404) 894-3110	22c OFFICE SYMBOL ASQB-GI

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

The research herein was performed for the Army Institute for Research in Management Information, Communications, and Computer Sciences (AIRMICS), the RDTE organization of the U.S. Army Information Systems Engineering Command (USAISEC). The sponsor for the project was the Office of the Director of Information Systems for Command, Control, Communications, and Computers (ODISC4). The principal investigator was Dr. H. Dunsmore of Purdue University.

Accession For	
NTIS CRA&I	✓
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

THIS REPORT HAS BEEN REVIEWED AND IS APPROVED

s/ [illegible]
Glenn E. Racine
Chief
CISD

s/ [illegible]
John R. Mitchell
Director
AIRMICS

Software Development Information Supported by Typical CASE Tools

S. Varnau
H. Dunsmore

Software Engineering Research Center (SERC)
Department of Computer Sciences
Purdue University, West Lafayette, IN 47907

SERC-TR-77-P
July, 1990

Technical Report 3.1 from the Research Project:
Evaluation of DCDS for Meeting the Data Collection
Requirements for Software Specification, Development, and Support

Abstract

In this study we consider that the information collected by a CASE tool is its most important feature. In this work we are identifying data collection requirements for a fully flexible CASE environment. We have compiled a preliminary list of information needs for CASE tools and have compared this list to the information collected in five existing products and desirable functionality as suggested by Henderson and Cooprider [HEND88].

We include detailed definitions and discussion of the identified information requirements. These requirements are divided into two categories. **Product** data includes everything which describes the software product itself. **Process** data includes everything which reflects the activity involved in developing and supporting the product. Product data is further subdivided into **description**, **implementation**, **verification**, and **maintenance** categories. Process data is subdivided into **management**, **coordination**, and **quality control** categories.

In our assessments we found that the representative CASE tools and Henderson and Cooprider's report score very well in the Product Description categories. Henderson and Cooprider's report contains functions that include most of the useful information in the Product Implementation categories, but the representative CASE tools do not score as well. The Product Verification and Product Maintenance categories of information are supported very poorly by the representative CASE tools as

well as by Henderson and Cooprider's report. The results in the Process Management categories are variable. Process Coordination is covered quite adequately by Henderson and Cooprider's functions. The representative CASE tools are adequate for only 2 of the 5 Process Coordination categories. Process Quality Control is covered better by the Henderson and Cooprider report than by the representative CASE tools. No tool is so much as adequate in any of these categories.

Background

Discussion of Approach

Many CASE products are on the market with varying capabilities for supporting the analysis, design, coding, and testing of software [GANE90]. This indicates the complexity of the issue at hand. How does one go about choosing (or constructing) a software development environment complete enough to enhance the quality and productivity of all phases of a particular software project, and flexible enough to handle various types of projects? Indeed, a Department of Defense manual on the subject [CRAW88] enumerates six different perspectives of programming support environments:

1. A Collection of Tools

2. A Methodology Support System

3. An Information Management System

4. An Interactive System

5. A Knowledge-Based Expert System

6. A Stable Framework (for long-lived, evolutionary systems)

While conceding that many other perspectives surely exist, we add three more to this list:

7. A Project Manager (aids in scheduling, budgeting, work assignments)

8. A Coordination System (enhances communication and coordination among the software developers)

9. A Configuration System (controls versions and revisions of specifications, design, and/or code)

Several methods for evaluating these products have been proposed (for example, [BARA89, DUNS87, HEND88]). The most obvious and frequently used method of evaluating a software engineering environment is to study its functionality [BARA89, HEND88]. The pertinent questions are "What does it do?" and "How well does it do it?". This method covers all the above perspectives, and gives a good feel for our primary goals of software quality and productivity benefits. This functionality viewpoint, however, usually sacrifices the goal of flexibility. Implicitly, functionality evaluation limits what technology is considered according to the list of requirements adopted. To take future technology into consideration, such a list must be prophetic (at best) or vague (at worst).

Since the software engineering field is constantly evolving, we are evaluating CASE environments on the basis of the three goals: **quality**, **productivity**, and **flexibility**. As a first requirement, we describe a narrower view of CASE environments, which is inherently adaptable to differing and evolving needs.

A CASE environment should not be merely a workbench of independent tools, but *a set of integrated tools, using a consolidated collection of information* [FISC89]. A consolidation (i.e., centralization) of project data avoids redundancy, enhances communication, and encourages consistency. Tools which use that information may be added or replaced without affecting the core of the environment. As new technology becomes available, it may be integrated.

It is becoming increasingly common for software environments to have an **open architecture**. That is, there is a database of software project information that is accessible by various environment tools and perhaps even for perusal (and even modification) by the software developer. This is often achieved by custom translators or interfaces. Industry standards are currently being developed which will decrease reliance on such ad hoc methods. Most software environments currently encode software project information in a proprietary manner that is at least unintelligible and even unaccessible. We do note that some of the products we have seen (including Teamwork, Excelerator, and DesignAid included below) have varying degrees of semi-open architecture, but none really have a completely open architecture. However, as the industry progresses (as it seems to be doing) into open architecture systems, this will help immensely toward the goal of flexibility. Functionality in such an environment will be allowed to evolve and specialize. This suggests functionality will then be (if it is not already) a secondary concern, and that is why we have developed a new evaluation method.

From this new viewpoint, **the most important feature of a CASE environment is the information it collects**. In our work we are identifying data collection requirements for a fully flexible CASE environment. This information should be available to support all useful tools. This includes tools in common use and those that are still theoretical. The information requirements presented here do not depend on any particular representations, nor support any specific functions. Freedom from explicit representations and functions allows for unbiased appraisal of competitive environments.

To identify required data categories, we first compiled a preliminary list from our experience and from the related literature. Most of the current literature takes a functional viewpoint, so the data required for such functions had to be inferred. Next, this list was compared to the information collected in five existing CASE products and desirable functionality as suggested by Henderson and Cooprider [HEND88]. We used five CASE products which have already been evaluated as part of the SEES project [DUNS87]. This exercise was not intended to be an evaluation of these sources, but rather a way to compare and contrast information collected, to identify further data categories to be included in the list, and to assess the current state of practice in this area.

The information on the CASE products was gathered from their documentation and from limited use. By relying principally on how the product was intended to be used along with experimenting directly with the product, it was possible to determine full sets of information collected and intended to be collected. In this task, we were not interested in how well the information is used, or any other aspect of the product. A drawback of this method is that we were very dependent on each product's documentation. Some of the manuals were organized in a manner very conducive to the task at hand; others decidedly were not. Other sources were also used whenever available (e.g., [GANE90]).

The products examined:

DesignAid version 3.55, one of the CASE 2000 tools for Computer Aided Software Engineering available from Nastec Corporation, 24681 Northwestern Highway, Southfield, Michigan 48075.

EPOS version 3.3.0 - Engineering and Project-management Oriented Support system, available from Software Products & Services (SPS), 14 East 38th Street, New York, New York 10016.

Excelerator/RTS version 1.8A, available from Index Technology Corp., One Main Street, Cambridge, Mass 02142.

SA Tools version 1.00, available from Mentor Graphics, 8500 Southwest Creekside

Place, Beaverton, Oregon 97005.

Teamwork version 3.0, available from Cadre Technologies, 222 Richmond Street, Providence, Rhode Island 02903.

Also included in this investigation are the functions included in Henderson and Cooprider's model of CASE technology [HEND88]. Inferences were made to discern what information is needed by those functions.

The most difficult type of information to capture and use are relationships among other data items. Often relationships are inherent within data, data formats (such as naming schemes), and heuristics in functional tools. The categories listed are not necessarily mutually exclusive. Some information could legitimately be put into more than one category.

Sometimes the scope of CASE products comes into question. Some environments include support for system engineering (including hardware along with software considerations) or project management. Since these aspects interact with traditional software aspects in very important ways, we have taken a broad view of software engineering.

Some information may be derivable from other collected information (e.g., metrics). This type of redundancy may very well be desirable to avoid undue processing time. Some information may be derivable from external sources (e.g., e-mail addresses). This type of redundancy may not be desirable to avoid unnecessary use of storage and unnecessary development of new tools which duplicate existing ones. External resources may be used through specific tools, or peripheral databases may be useful. These types of issues are implementation-dependent. The point is that all of the data items listed should be available to the CASE environment and to a complete set of integrated tools.

Results

Below are the results of this phase of our work. We have identified information collected and supported by representative CASE tools, the information which CASE tools should collect based on [HEND88], and (based on our experience and concurrent work) information that should be part of a software development environment that appears neither in the CASE tools nor in [HEND88].

We include detailed definitions and discussion of the identified information requirements. These requirements are divided into two categories. **Product** data includes everything which describes the software product itself. The typical results of a software project are the requirements, specifications, design, implementation, code metrics, test plans, etc. These materials comprise product data. **Process** data includes everything which reflects the activity involved in developing and supporting the product. This includes personnel, schedule, budget, etc.

Product data is further subdivided into **description**, **implementation**, **verification**, and **main ance** categories. Description data consists of information from the development phases commonly known as requirements, specifications, and design. This category of items serves as a plan for the product in the initial stages of a project and as documentation in later stages. Note that description data should be flexible enough to facilitate analysis and design of software, user documents, test plans, and anything else needed. Implementation data consists of the deliverable components of a product. This includes code and documentation for the end user. Verification data consists of correctness information (typically testing information). Maintenance data consists of information used in ensuring continuing usefulness of the project after initial delivery.

Process data is subdivided into **management**, **coordination**, and **quality control**. Management data is used to control the project in terms of time and resources used. Coordination data is used to help personnel communicate, thus increasing quality and productivity. Quality control data is used to ensure and generally support development of a correct, robust, safe product.

Another term which appears in this report is *component*. This is a general term referring to an element of unspecified type or a group of elements. A component usually refers to a part of the deliverable product (e.g., a code module or a document). A component may also be part of a specification, design, etc. which refers to code or documents.

Because of the great variety of software applications and the great variety of perspectives on software, this list is necessarily somewhat general. The definitions and discussion, however, provide pertinent subcategories. We have attempted to make this list as concrete as possible. That is a difficult task in the frontier area of software engineering. While we were generally surprised by how many of these requirements were met by at least one of the CASE tools considered, some in the areas of maintenance, verification, coordination, and quality control were conspicuously absent.

PRODUCT

Product Description - Planning, development, documentation of all aspects of the specific product. This is the major category that includes most of what we think of when we think of what the software does.

Functionality - What the product must do. This information should reflect the requirements and specifications for the software. It can be in a formal, semi-formal, or just a natural language format. It should include data input, data output, product behavior, and other properties such as portability and security.

Interfaces - Interaction with external systems. This information should detail what external systems are related to this software and the specific types of interactions between the software and the external systems.

Performance - Time and space that the product uses. This is information that describes the required memory and disk space for the software, along with standard (or typical) execution times. The information may be quite complicated if the software can be run in various size configurations or if execution times are varied dependent on input parameters.

Time Constraints - Real time limitations. This information outlines the time performance constraints placed on the software. This includes any partial or total constraints placed on execution times.

Fault Tolerances - Error and failure handling. This information outlines the acceptable responses of the software to "erroneous" input or to hardware failure. Such errors and failures can include exceptions, faults, and resource limitations. The information in this category can include the types of error messages that are to appear, the kinds of errors that need not be detected, and the kinds of recoveries expected from certain errors.

Data Flow - Movement of data in and out of components or stores. This information describes the way in which data moves throughout the software. It treats each component or store as a data-handling entity and describes that information that moves in and out of that entity (including what the data is, where it came from, and where it is going).

Process Flow - Execution progression of components; sequential/parallel. This information describes the software from a control flow viewpoint discussing the flow of execution in both normal and abnormal situations. It also includes sequential and parallel control flow information.

Resources - Resource usage of component; hardware considerations. Resources are any entities external to the software. This information discusses resources that either supply information to the software or receive information from the software.

Structure - Static decomposition of components. This information conveys any logical grouping of components for any reason; for example, grouping all components that deal with the same database. There may be several static decompositions for the same software.

Entity-Relationships - Relationships among components and externals. This information includes all of the typical E-R type information, e.g., for each entity to what other entities is it related and in what manner.

Communication - Internal interfaces. Within the software how is communication accomplished? What messages (in the object-oriented sense) are communicated among the software's entities?

Data - Often now being called the **Data Dictionary**, **Data Encyclopedia**, or **Data Repository**. Data types, operations, constants, descriptions, stores, relationships, objects and classes, processes, data flows, events, states, external entities. May be related to Project Index data (see Process Coordination).

Requirements/Design - Relationships of goals and components. This information tells which requirements are related to (satisfied by) which elements of the design.

Design/Performance - Relationships of structure and performance. This information tells which elements of the design are related to the various performance constraints.

Description/Implementation - Relationships of planned and implemented components. This information links the requirements and specifications (description of the software) with the actual implementation. That is, what components implement the requirements and specifications.

Design/Design - Relationships of alternative design representations. For software with more than one design proposed, how does each relate to the other? What are the functionality and performance tradeoffs of each?

Prototypes - What prototyping activity is planned? What specific aspects of the software is to be prototyped? What will be done with the prototype? What simulations will be conducted? What experiments will be tried to test requirements, specifications, design, etc. This information, when complete, should include the prototype goals (questions the prototype is designed to answer) and results (experimentally-determined answers), as well as the actual prototype product, simulation code, etc.

Product Implementation - This is the major category that includes the actual software product (i.e., code, documents, etc.) as well as relevant information.

Actual Product - Code, Documents for end user. This is the software and documentation produced. It consists of all new (and possibly re-used) code and the text and graphics necessary to produce documentation for the software. This category is closely related to Configuration (see Process Coordination) which keeps track of versions, revisions, etc.

Metrics - Product statistics. This information consists of any and all metrics computed primarily from the software code (but possibly also from documentation or other related representations of the product). It may include (but is not limited to) such metrics as lines-of-code, size of data structure, and complexity (e.g., v(G)). Such metrics may be used for management, testing, maintenance, performance, and even quality control purposes.

Library - Globally available, re-usable components. This information contains either actual re-usable components (or some sort of pointer to them) that will be (or have been) employed in the implementation of the software. Such a library may have project, company, or even wider scope.

Templates - Outlines and examples of common components. This information contains sample components that conform to project, company, or wider standards. Such components may simply be bare-bones schema with little actual code or may be nearly complete components that require only minor modification before use in the software.

Compile Parameters - How code is compiled for testing, debugging, and (ultimately) for generating a production version. This information includes standard compilation parameters, ways of testing various versions, searching order for external components (such as re-used components), and special parameters necessary for preparing the product version.

Product Verification - This is the major category that includes all information related to testing the software (or any related activity that attempts to discover and correct errors).

Test Plan - Outline of testing process. This contains at least rudimentary information about how the software is to be tested: what types of testing procedures (perhaps formal methods) will be pursued, what tools will be used, what types of test data, what will be done about errors that are discovered, etc.

Test Tools - Custom functions for debugging and testing. This is information about the specific tools that will be (were) used for testing the software. These can include tools that are part of the CASE tool, standalone external tools, or specific test harnesses to be produced as part of the software development process.

Test Suites - Test data and expected results. This information describes specifically how test data is to be generated, how the software is to be "exercised" with this data, and how the results are to be interpreted.

Status - This information (collected during the software testing process) outlines which tests have detected the presence of an error and which tests have failed to detect the presence of an error. Obviously, it is possible to tell from this information which tests have been run (and either detected or failed to detect errors) and which tests have not been run. For regression tests, this information will tell which have been run on which versions and which revisions.

Errors Found - Errors discovered through testing; error reports. This information outlines what errors have been discovered, which have been corrected, which are planned to be corrected, and which (if any) are not planned to be corrected.

Verification/Description - This information links the requirements and specifications (description of the software) with the verification process. That is, what has been (will be) done to assure that specific requirements and specifications have been tested.

Analysis - Results of matching implementation against description (i.e., requirements and specifications). This information includes such items as types of errors, time and space performance, error and failure handling, consistency, and completeness.

Product Maintenance - This is the major category that includes all information related to the maintenance of the software, its upkeep, and support of the product in use (and perhaps even in late development stages).

Maintenance History - This information includes all actual changes made and known problems not yet corrected. It also includes information about various software releases and versions and how they differ from each other.

Special Cases - How the product is being used. How the product is being customized. This includes any release or version related information not included in the Maintenance History sub-category above due to special circumstances.

Complaints - Reported errors and their locations, problems; evaluations; replies. This information includes all requests for changes to the software based on actual errors (i.e., the software fails to meet one of its requirements).

Proposed Changes - Reported desires for new versions (including specific modifications); evaluations; replies; planned upgrades. This information includes all requests for changes to the software based on enhancements (i.e., the software meets its requirements, but it could do something even more useful for the end user).

General Information - Any other information related to the software as it is in operation; for example, (but not limited to) market penetration, customer addresses and contacts, and versions and licenses.

PROCESS

Process Management - Resource management for the software project. This is the major category that includes most of the management information pertaining to the software development process. A good CASE tool should support most of the information maintained and manipulated by good stand-alone project management tools.

Schedule - Time to finish each task. This information will include both estimates of task durations and triggering mechanisms (for those not yet completed) as well as actual start, stop, and duration times (for those tasks already completed). It will include any relevant dependency and status information, as well.

Budget - This information includes estimates of salaries, personnel costs, hardware costs, etc. (for tasks not yet completed) as well as actual salaries, personnel costs, and hardware costs (for those tasks already completed). It will include any relevant dependency and status information, as well.

Personnel Assignments - This information includes responsibilities (who is responsible for each aspect of the software development), backups (who are available to step in for those with primary responsibilities), authorities (who has read/write access to what project data), as well as individual data (experience, skills, etc.) for each member of the software development team.

Environment Customization - This information describes the environment in which this project is being developed (including how it may differ from the standard software development environment in this company). What procedures, tools, techniques, languages, management standards, coding standards, and documentation standards are being used. How text and graphics are formatted for various media. This information outlines how the software is to (does) interact with the end users. Information such as standard screen formats, standard error formats, standard input "forms" are all included in this information.

Format Parameters - Parameters for input to and output from the CASE system, including reports throughout the software life cycle that keep management informed of the progress on this software project. What reports are to be generated, what schedule is to be followed for them, are they to be manually or automatically generated, how should they look for various media.

Process Plan - What plan is to be (was) followed in developing the software. What phases are to be employed, what standards, and overall schedule. This can even include pre-project bidding and contracting information and some allowance for process improvement.

Process Coordination - This major category includes all information needed by the software development team for cooperation, communication, and organization.

Project Directory - Project, company, or environment scope directories. This information includes all linkages to people, requirements, specifications, design, code, and testing relevant to this software project. For example, in the people category it can include all personnel working on the project, personnel with previous experience on this or a similar project, personnel with consulting capabilities outside the project, etc.

Configuration - Arrangement of all product and some process data. This includes such information as (but is not limited to) software versions, revisions (history of the software), structural relationships, and control locks (overwrite protection).

Standards - Project consistency rules. This information includes all standards that are to be (were) followed during software development. Note that several other categories include some standards. In this category they are to be all collected including documentation (perhaps the most important), personnel, design, coding, messaging, and implementation standards.

Communication - Intra-group communication. This information includes names, addresses, phone numbers, e-mail addresses, and office locations of all personnel working on the project. It can also include (but is not limited to) mail aliases (mailing lists), note logs, meeting minutes, note/component relationships (i.e., topical index for notes, references).

Communication Formats - Idea communication media. This includes information on the various modes of communication among software development team members: for example, (both in-person as well as electronic versions of the following) forums, bulletin boards, brainstorming sessions, votes, etc.

Process Quality Control - This major category includes all information pertaining to quality assurance including product quality, process quality, run-time environments, and history.

Quality Goals - Criteria to measure quality. This includes information from requirements, specifications, and otherwise that can be used to assess the quality of the completed software project.

Fault Consequences - What happens if the product fails. This information describes the severity of the problems involved if the entire product or any components thereof fail to operate according to expectations.

Target Environment - How will the product be used. The software must operate within certain hardware and software constraints. This includes such information as the type of operating system, LAN operation, possible abuses, etc.

Inspections - Standards, schedules, participants, results. This includes information about what inspections are planned (or for a completed project, what inspections were conducted). It also includes information on classes, design meetings, problem resolution meetings, and informal meetings.

User Input - Customer/End-user evaluations and comments. What user input is going to be (was) collected. How is it to be used. What effect will it have on the developing and completed software product. What input will it obtain from experts in the field.

References - Miscellaneous, external references. This can include (but is not limited to) references to similar projects, projects in the same application area, projects conducted for similar hardware systems, projects developed by the same or similar software development teams, etc.

Project History - Record of changes and results of the process. This information includes all aspects of project history that is not found in Configuration (see Process Coordination). It may include (but is not limited to) project summaries, post-mortem analyses, process evaluations, and process improvement suggestions.

The sources used in this investigation were detailed above. Since some of the categories are broad and include subcategories, the following scale is used to show how completely the requirement was met. Remember that the Henderson/Cooprider model is a functional "wish list", not an implemented product.

- == No support at all or not addressed by tool (equivalent to 0)
1 == Possible to incorporate information, but not specifically supported
2 == Category addressed, but not fully supported
3 == Adequate
4 == Exceptional treatment of category
5 == Could not be better

Note that just because two items receive the same number for the same category, this does not mean that they are functionally equivalent for that category. For example, both EPOS and Teamwork rate a 3 for the category Interfaces below. This does not mean that EPOS and Teamwork have identical methods by which the developer can describe software interaction with external systems - only that we consider the Interface methods used by both EPOS and Teamwork as "Adequate".

PRODUCT - Description

Category	Henderson/ Cooprider	EPOS	Teamwork	Excelerator	DesignAid	SA Tools
Functionality	3	3	4	2	2	2
Interfaces	2	3	3	4	2	2
Performance	3	2	-	4	-	-
Time Constraints	1	3	1	3	-	-
Fault Tolerances	1	2	-	2	-	-
Data Flow	3	3	4	3	3	3
Process Flow	3	4	3	3	3	-
Resources	1	3	-	3	-	-
Structure	4	3	3	3	4	1
Entity-Relation.	2	2	3	3	3	1
Communication	3	3	3	4	3	2
Data	3	3	3	3	3	2
Req./Design	3	3	-	-	-	-
Design/Perf.	3	1	-	2	-	-
Descrip./Impl.	4	3	2	1	1	1
Design/Design	1	3	2	1	-	-
Prototypes	2	-	-	2	-	-
Mean Scores	2.5	2.6	1.8	2.5	1.4	0.8
Range	1-4	0-4	0-4	0-4	0-4	0-3
Inadeq. Pctage.	41%	29%	53%	41%	65%	94%

Note that the *Mean Scores* average the ratings for each tool (as well as the Henderson and Cooprider report) for all categories. The *Range* gives an idea of the variability of that tool (report) across all categories. The *Inadeq. Pctage.* line reports the percentage of all ratings for each tool (report) that are 0, 1, or 2 (instead of 3 or 4 - there were no 5's) and thus judged to be *inadequate.*

The representative CASE tools do not score particularly well in the Product Description categories. This is somewhat surprising, since most of the stated goals of such tools fall squarely in these categories. Notice that the functions included in Henderson and Cooprider's model of CASE technology score about as well as EPOS and Excelerator.

EPOS appears to be the best of the five CASE tools we analyzed based on the Product Description categories. Its 2.6 mean is the highest and 29% inadequacy percentage is the lowest. Excelerator (2.5 and 41%) is comparable in performance. The fact that none of these means reaches 3 or 4 shows that there is substantial room for improvement.

PRODUCT - Implementation

Category	Henderson/ Cooprider	EPOS	Teamwork	Excelerator	DesignAid	SA Tools
Actual Product	3	3	1	1	1	-
Metrics	2	-	1	1	-	-
Library	4	1	2	2	-	-
Templates	3	-	1	1	3	-
Compile Param.	-	-	-	-	-	-
Mean Scores	2.4	0.8	1.0	1.0	0.8	0.0
Range	0-4	0-3	0-2	0-2	0-3	0
Inadeq. Pctage.	40%	80%	100%	100%	80%	100%

Our assessment is that Henderson and Cooprider's report contains functions that include a good deal of the useful information in the Product Implementation categories. The representative CASE tools do not score well at all (especially in the Metrics and Library categories). Finally, note that there is no information collection support for the Compile Parameters category. This category constitutes information that we believe should be part of a software development environment that appears neither in the Henderson and Cooprider report nor in the representative CASE tools.

Not any of the five CASE tools we analyzed seems anywhere near adequate for these categories as attested by the extremely low mean scores and very high inadequacy percentages.

PRODUCT - Verification

Category	Henderson/ Cooprider	EPOS	Teamwork	Excelerator	DesignAid	SA Tools
Test Plan	-	3	-	-	-	-
Test Tools	-	-	-	-	-	-
Test Suites	-	-	-	-	-	-
Status	-	-	1	-	-	-
Errors Found	1	2	-	-	-	-
Ver./Descrip.	-	2	-	-	-	-
Analysis	1	2	1	2	-	-
Mean Scores	0.3	1.3	0.3	0.3	0.0	0.0
Range	0-1	0-3	0-1	0-2	0	0
Inadeq. Pctage.	100%	86%	100%	100%	100%	100%

The Product Verification categories of information are supported very poorly by the representative CASE tools as well as by Henderson and Cooprider's report. EPOS has adequate support for Test Plan information, but there is no support at all for such categories as Test Tools and Test Suites. All in all, this category of software design information is supported very poorly by existing technology.

PRODUCT - Maintenance

Category	Henderson/ Cooprider	EPOS	Teamwork	Excelerator	DesignAid	SA Tools
Maintenance History	2	1	1	-	2	-
Special Cases	-	1	1	-	-	-
Complaints	-	3	-	-	-	-
Proposed Changes	-	3	-	-	-	-
General Information	-	-	-	-	-	-
Mean Scores	0.4	1.6	0.4	0.0	0.4	0.0
Range	0-2	0-3	0-1	0	0-2	0
Inadeq. Pctage.	100%	60%	100%	100%	100%	100%

EPOS has adequate support for the categories of Complaints and Proposed Changes. But, by and large, the Product Maintenance categories are supported very poorly by the representative CASE tools as well as by Henderson and Cooprider's report.

PROCESS - Management

Category	Henderson/ Cooprider	EPOS	Teamwork	Excelerator	DesignAid	SA Tools
Schedule	4	4	1	-	-	-
Budget	-	3	-	-	-	-
Pers. Assign.	4	3	1	1	1	-
Environ. Custom.	3	2	2	-	2	2
Format Parameters	3	-	-	4	3	-
Process Plan	-	-	-	-	-	-
Mean Scores	2.3	2.0	0.7	0.8	1.0	0.3
Range	0-4	0-4	0-2	0-4	0-3	0-2
Inadeq. Pctage.	33%	50%	100%	83%	83%	100%

The results in the Process Management categories are variable. Henderson and Cooprider's report scores well in every category except that it has no provisions for Budget or Process Planning types of information. EPOS supports Schedule, Budget, and Personnel Assignments fairly well. Both Excelerator and DesignAid support Format Parameters at least adequately. But, only EPOS comes anywhere close to adequacy for these categories.

PROCESS - Coordination

Category	Henderson/ Cooprider	EPOS	Teamwork	Excelerator	DesignAid	SA Tools
Project Direct.	3	1	3	3	3	-
Configuration	3	3	1	1	3	-
Standards	3	-	-	-	-	-
Communication	3	-	1	-	-	-
Commun. Formats	3	-	-	-	-	-
Mean Scores	3.0	0.8	1.0	0.8	1.2	0.0
Range	3	0-3	0-3	0-3	0-3	0
Inadeq. Pctage.	0%	80%	80%	80%	60%	100%

Process Coordination is a set of categories covered quite adequately by Henderson and Cooprider's functions because it is a focus of their report. The representative CASE tools are also variably adequate for the Project Directory and Configuration categories. But, the representative tools score very badly in the Standards, Communication, and Communication Formats areas. None of these CASE tools even approach adequacy for this set of categories.

PROCESS - Quality Control

Category	Henderson/ Cooprider	EPOS	Teamwork	Excelerator	DesignAid	SA Tools
Quality Goals	3	-	-	-	-	-
Fault Conseq.	-	-	-	2	-	-
Target Environ.	2	-	-	-	-	-
Inspections	-	-	-	-	-	-
User Input	2	1	-	-	-	-
References	3	-	-	2	1	-
Project History	1	1	-	-	-	-
Mean Scores	1.6	0.3	0.0	0.6	0.1	0.0
Range	0-3	0-1	0	0-2	0-1	0
Inadeq. Pctage.	71%	100%	100%	100%	100%	100%

Process Quality Control is again covered better by the Henderson and Cooprider report than by the representative CASE tools. No tool is so much as adequate in any of these categories, and Henderson and Cooprider's report only reaches adequacy in the Quality Goals and References categories. Fault Consequences, Inspections, and Project History are glaring weaknesses in this category.

SUMMARY

The table below summarizes the mean scores from the previous 7 tables:

Category	Henderson/ Cooprider	EPOS	Teamwork	Excelerator	DesignAid	SA Tools
Product Description	2.5	2.6	1.8	2.5	1.4	0.8
Product Implementation	2.4	0.8	1.0	1.0	0.8	0.0
Product Verification	0.3	1.3	0.3	0.3	0.0	0.0
Product Maintenance	0.4	1.6	0.4	0.0	0.4	0.0
Process Management	2.3	2.0	0.7	0.8	1.0	0.3
Process Coordination	3.0	0.8	1.0	0.8	1.2	0.0
Process Quality Control	1.6	0.3	0.0	0.6	0.1	0.0
Mean Scores	1.9	1.6	0.9	1.2	0.8	0.3

Note that the mean scores suggest that none of the 5 CASE tools we examined can be considered adequate based on the categories of information we analyzed. There is some variability among the CASE tools, but EPOS and Excelerator appear somewhat better based on our assessment. However, the reader should not put too much stock in these unweighted averages. More important to some readers might be to use these component scores to construct a more meaningful weighted average.

Finally, it is interesting to note that the best performance in our assessment was achieved by the Henderson/Cooprider report. But, this report has the advantage that it is only a "wish list" of desirable components of a CASE system. The authors made no attempt to implement such a system.

Our research to this point suggests that the current state-of-the-art in CASE technology is not adequate to provide the kind of software development support needed to meet current data collection requirements for software specification, development, and support.

References

[BARA89] Baram, Giora. "Selection Criteria for Analysis and Design CASE Tools". *ACM SIGSOFT Software Engineering Notes,* October 1989, 73-80.

[COOP89] Cooprider, Jay G. and John C. Henderson. "Perspectives on the Performance Impacts of Technology on the Prototyping Process". Center for Information Systems Research Technical Report. MIT Sloan School of Management. 1989 (in press).

[CRAW88] Crawford, Bard S. and Peter G. Clark. *Evaluation and Validation (E&V) Reference Manual.* AFWAL/AAAF, Wright-Patterson Air Force Base, Ohio. Version 1.1, October, 1988.

[DUNS87] Dunsmore, H. E., W. M. Zage, D. M. Zage, and G. Cabral. "A Tool for Evaluating Software Engineering Environments". Software Engineering Research Center Technical Report TR-2-P. June, 1987.

[FISC89] Fischer, Kurt. "The Software Factory of the Future". *CTContact.* 2,3 May, 1989.

[GANE90] Gane, Chris. *Computer-Aided Software Engineering.* Prentice-Hall, Englewood Cliffs, NJ, 1990.

[HEND88] Henderson, John C. and Jay G. Cooprider. "Dimensions of I/S Planning and Design Technology". Center for Information Systems Research Technical Report. MIT Sloan School of Management. September, 1988.

[HUMP87] Humphrey, Watts S. and William L. Sweet. "A Method for Assessing the Software Engineering Capability of Contractors". Software Engineering Institute Technical Report CMU/SEI-87-TR-23. Fall, 1987.